N-body Project

Yifei Gu

260906596

The workflow of my simulator:

For Step 1) **Get the potential field of a single particle**

A 3D grid is generated by *np.meshgrid* and the grid is centered to the particle. Then the potential field of the single particle (kernel) is calculated by Green's function. **For non-periodic boundary condition**, I will use **twice number of cells for the grid**.

```
def greens(n,soft,G,non_periodic):
    # n is the number of grids
    if non_periodic:
        n=2*n
    x=np.arange(n)/n
    x[n//2:] = x[n//2:]-n/n
    xmat,ymat,zmat = np.meshgrid(x, x, x)
    kernel = np.zeros([n,n,n])
    dr=np.sqrt(xmat**2+ymat**2+zmat**2+soft**2)
    kernel=1*G/(4*np.pi*dr)
    return kernel
```

For Step 2) **Get the potential field for multiple particles**

We need to setup a grid for the density field and smear the greens function through the density filed to get the entire potential field of multiple particles. Here is how I defined my function to get density field:

```
def density(position,n,m):
    grid_min = 0
    grid_max = n
    den, edges = np.histogramdd(position,bins=(n,n,n),
                                range=((grid_min, grid_max),
                                       (grid_min, grid_max),
                                       (grid_min, grid_max)),
                                        weights=m)
```

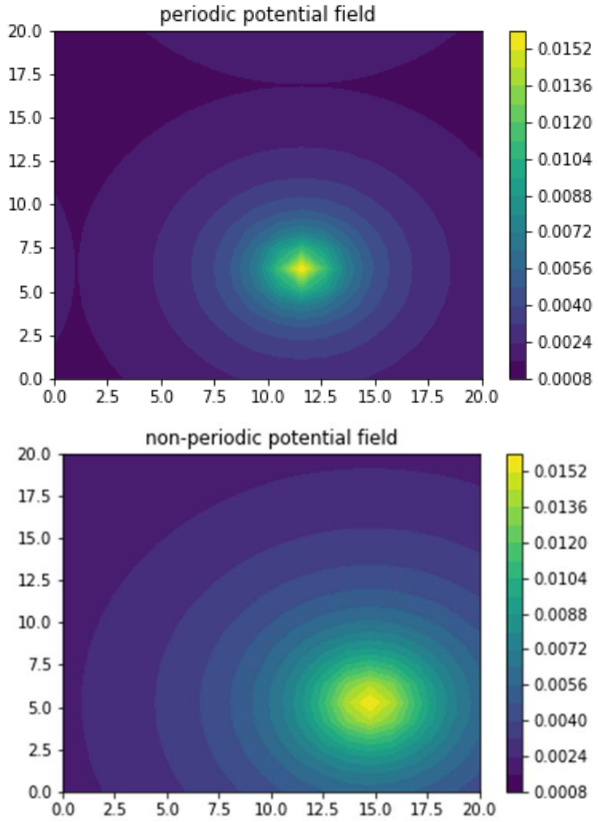
To do the convolution, we need to shift the kernel so that it is aligned to the density grid. Then the kernel can be transformed to the frequency space.

```
kernel = greens(n,soft,G,non_periodic=non_periodic)
kernel=np.fft.fftshift(kernel)
kernelft=np.fft.rfftn(kernel)
```

Now the outputs of greens function and density function can be used to get the potential field. **For the non-periodic condition**, I did **zero padding** to twice number of cells as before and I will cut it back to the number of cells as before after the inverse FFT. The particles that go off the grid will have zero potential.

```
def get_pot(kernelft,den,non_periodic):

    pot=den.copy()
    if non_periodic:
        pot=np.pad(pot,(0,pot.shape[0]))
    potft=np.fft.rfftn(pot)
    pot=np.fft.irfftn(potft*kernelft)
    pot=np.fft.fftshift(pot)
    if len(den.shape)==3:
        pot=pot[:den.shape[0],:den.shape[1],:den.shape[2]]
    return pot
    print("error in get_pot - unexpected number of dimensions")
    assert(1==0)
```

Here is the comparison of the potential field between periodic condition and non-periodic condition:



For Step 3) **Get the force by take derivatives of the potential field**

```
def get_forces(pot,n):
    force = np.gradient(pot,1/n)
    force_x = np.asarray(force[0])
    force_y = np.asarray(force[1])
    force_z = np.asarray(force[2])

    return force_x,force_y,force_z
```

For Step 4) **Take steps with leapfrog algorithm**

With periodic condition, I will put back the particles go off the edge simply by taking the reminder. With non-periodic condition, the particles that go off the grid will disappear. The force grid is properly aligned to

the density grid so the correct force is apply for each particle at each step.

Here is the code:

```
def take_step(position,v,dt,n,kernelft,m,non_periodic):
    pos=position+0.5*v*dt

    if non_periodic==False:
        pos[pos<=0] = pos[pos<=0]%n
        pos[pos>=n]= pos[pos>=n]%n

    den = density(pos,n,m)[0]
    pot = get_pot(kernelft,den,non_periodic)
    force_x,force_y,force_z = get_forces(pot,n)

    #find the corresponding index in the force matrix for each particle
    bins = np.arange(0,n)
    f = np.zeros(position.shape)
    ind_x = np.digitize(position[:,0],bins,right=True)
    ind_y = np.digitize(position[:,1],bins,right=True)
    ind_z = np.digitize(position[:,2],bins,right=True)

    fx = np.zeros(position.shape[0])
    fy = np.zeros(position.shape[0])
    fz = np.zeros(position.shape[0])

    for i in range(position.shape[0]):

        fx[i]=force_x[ind_x[i]-1,ind_y[i]-1,ind_z[i]-1]
        fy[i]=force_y[ind_x[i]-1,ind_y[i]-1,ind_z[i]-1]
        fz[i]=force_z[ind_x[i]-1,ind_y[i]-1,ind_z[i]-1]
    f = np.c_[fx,fy,fz]
    #update the velocity and position
    vv=v+0.5*dt*f
    position=position+dt*vv
    if non_periodic==False:
        position[position<=0] = position[position<=0]%n
        position[position>=n]= position[position>=n]%n
    v=v+dt*f
    return position,v,pot,f
```

For Step 5) Tracking energy

```
def get_energy(position,v,pot,m):
    PE = -0.5*np.sum(pot)
    vx = v[:,0]
    vy = v[:,1]
    vz = v[:,2]
    KE = np.sum(0.5*(vx**2+vy**2+vz**2)*m)
    energy = PE+KE
    return energy,PE,KE
```

For Step 6) Parameter setting

The following all tasks in next sections are done with the same functions described above but with different parameter setting. Here is my section for setting up the parameters.

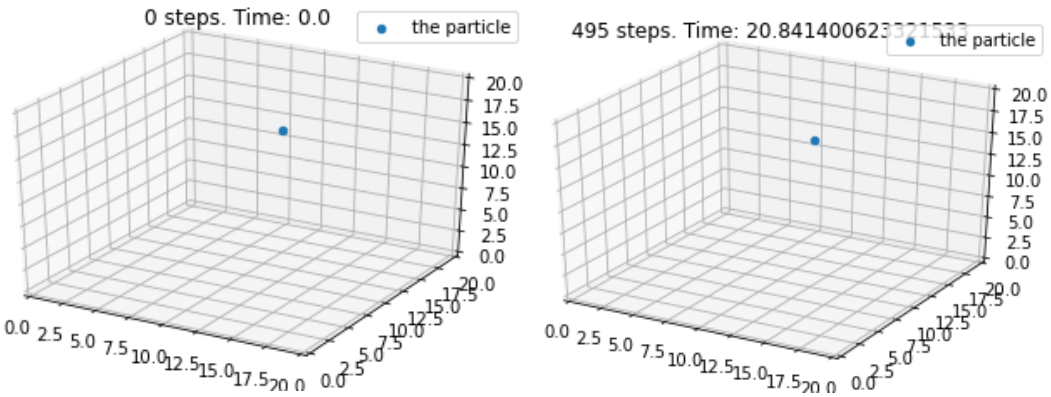
Task 1) One particle at rest

My parameter setting:

```
#grid number
n=20
#particle number
N=1
#mass
m=np.ones(N)
#random scattered initial position
position=n*np.random.rand(N,3)
#starting at rest
v=0*np.random.rand(N,3)
#time-step
oversamp=5
dt=0.04#0.04
#softening
soft=0.5
#gravational constant
G=0.01
#iterations
steps=100
#boundary condition
non_periodic=True
```

One particle starts at a random position and with 0 velocity stays at rest

Here is the image at the beginning and the end:

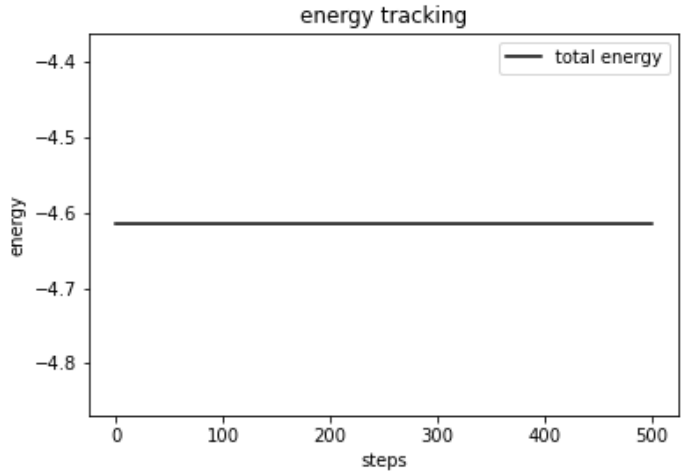


The position does not change:

```
The starting psotion of the particle: [[13.82250935  7.95426404 18.2515768 ]]
```

```
The ending psotion of the particle: [[13.82250935  7.95426404 18.2515768 ]]
```

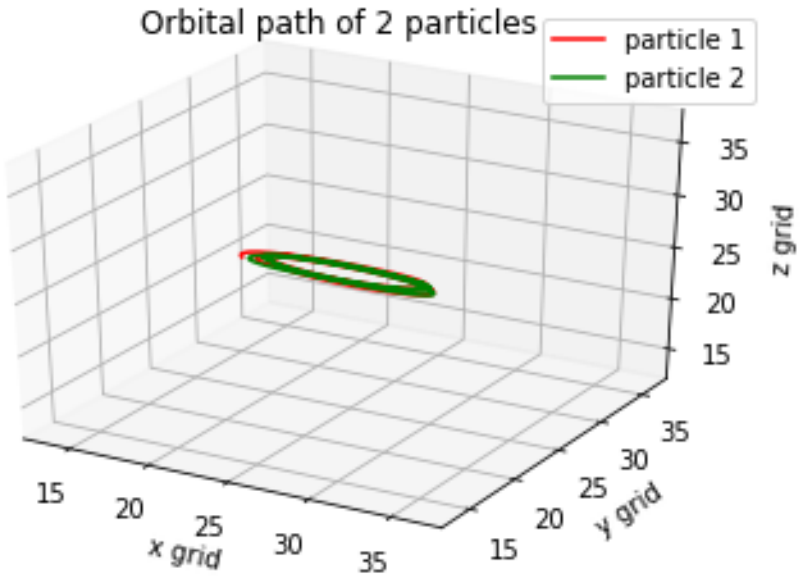
The energy stays conservative:

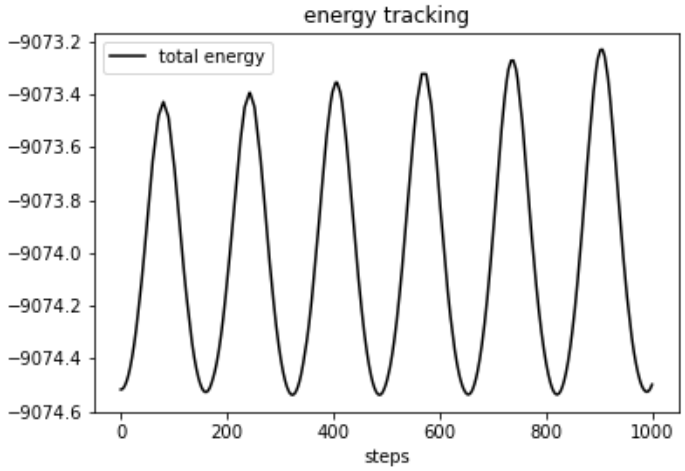


Task 2) Two orbiting particles

```
# ================================================================================
# setting parameters
# ================================================================================
#grid number
n=50
#particle number
N=2
#mass
m=np.ones(N)
#initial position
position = np.zeros((N,3))
position[0,0]=n*0.4
position[1,0]=n*0.6
position[:,1]=n/2
position[:,2]=n/2

#initial velocity
init_v=0.2#0.2
v=np.zeros((N,3))
v[:,1]=0
v[:,0]=0
v[0,2]=-init_v
v[1,2]=init_v
#time-step
oversamp=10
dt=0.1#0.04
#softening
soft=0.3
#gravational constant
G=0.5
#iterations
steps=100
#boundary condition
non_periodic=False
```

Two particles starting at the center of the grid with an opposite velocity along the Z axis.





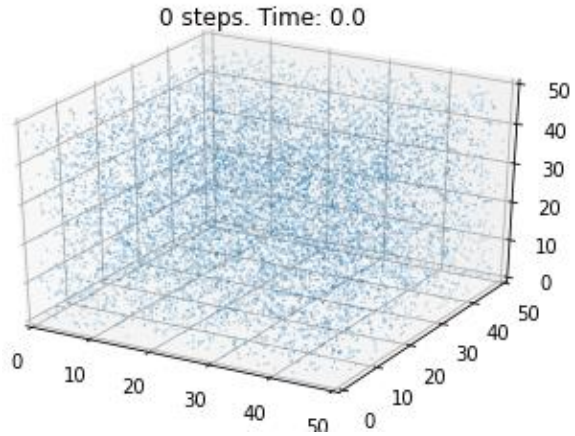
Two particles continue orbit with each other through the entire 1000 steps. The total energy is oscillating but within a small range.

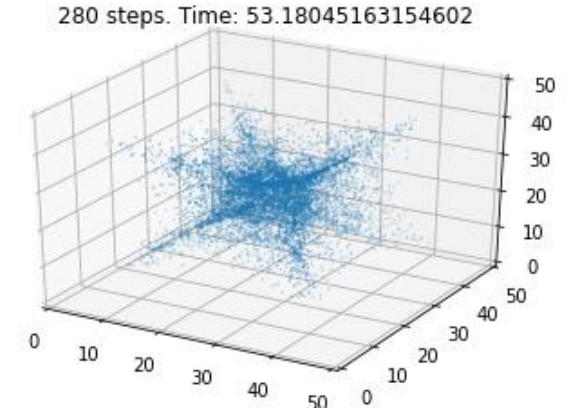
Task 3) N-body

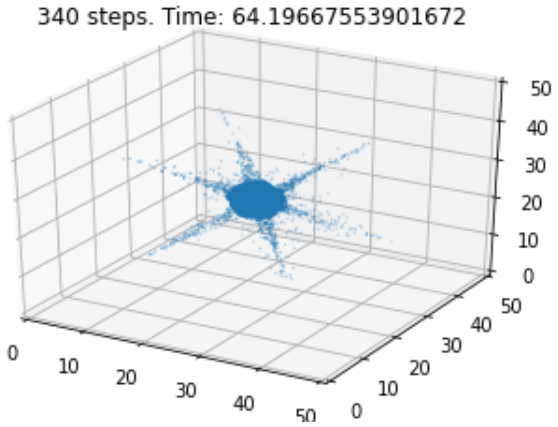
I generated 10000 particles scattered randomly in the grid with 0 velocity. Here are the parameters:

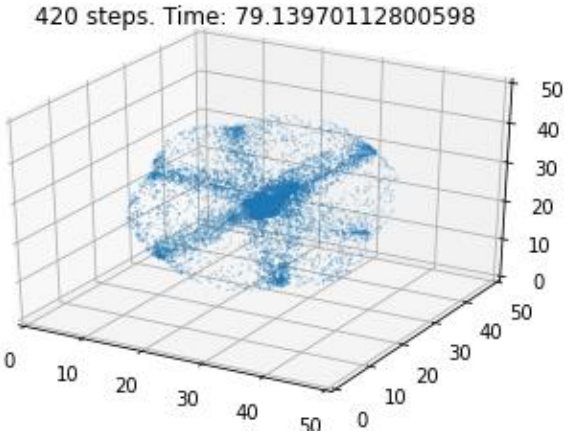
```
# ================================================================================
# setting parameters
# ================================================================================

#grid number
n=50
#particle number
N=10000
#mass
m=np.ones(N)
#random scattered initial position
position=n*np.random.rand(N,3)
#starting at rest
v=0*np.random.rand(N,3)
#time-step
oversamp=20
dt=0.01#0.04/0.01
#softening
soft=0.03
#gravational constant
G=0.001#0.01
#iterations
steps=50
#boundary condition
non_periodic=True
```

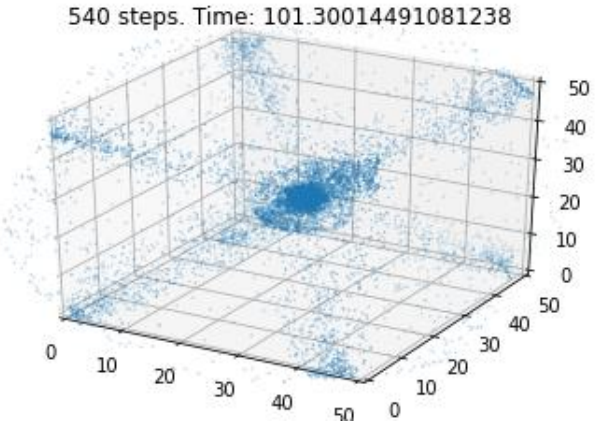
With non-periodic boundary condition:

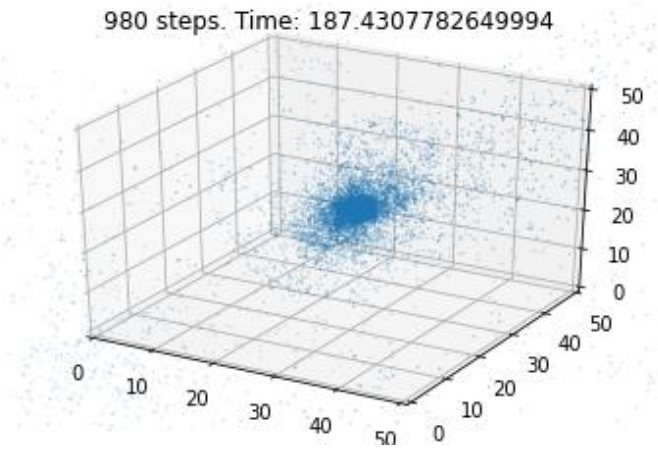




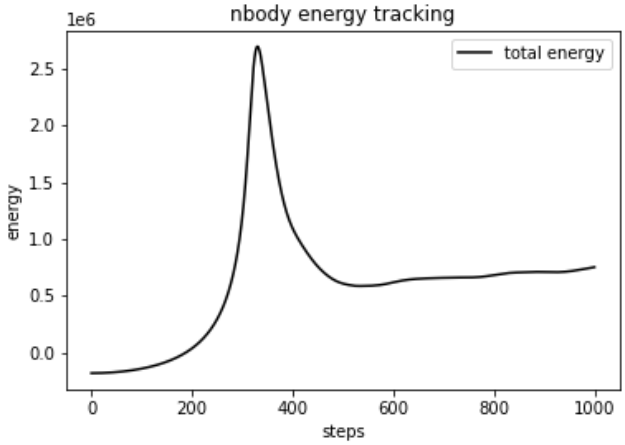

340 steps. Time: 64.19667553901672
0
10
20
30
40
50


420 steps. Time: 79.13970112800598
0
10
20
30
40
50

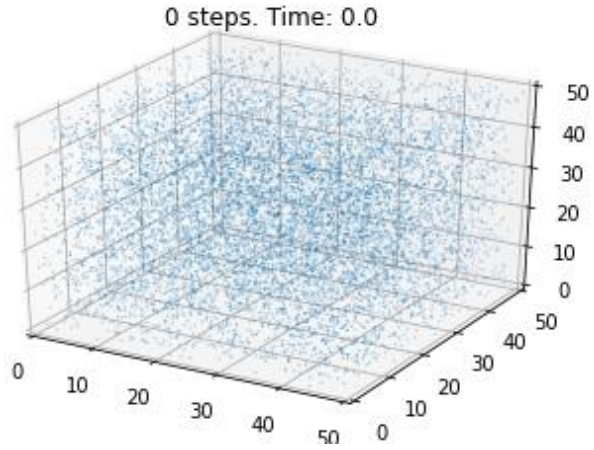

540 steps. Time: 101.30014491081238
0
10
20
30
40
50

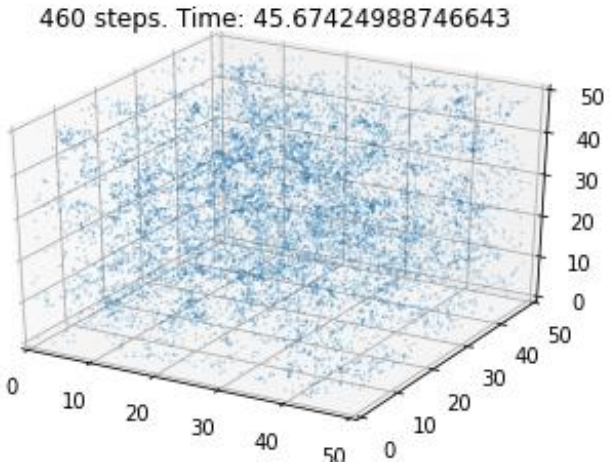

980 steps. Time: 187.4307782649994
0
10
20
30
40
50

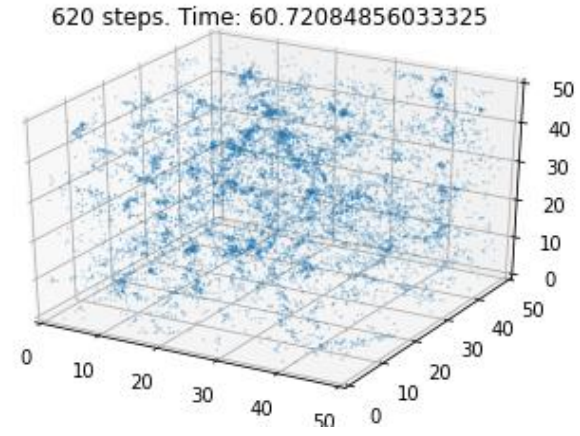
The particles form a ball shape. The energy does not conservative well. It blows up very quick and comes back down because of particles that fell off the edges.

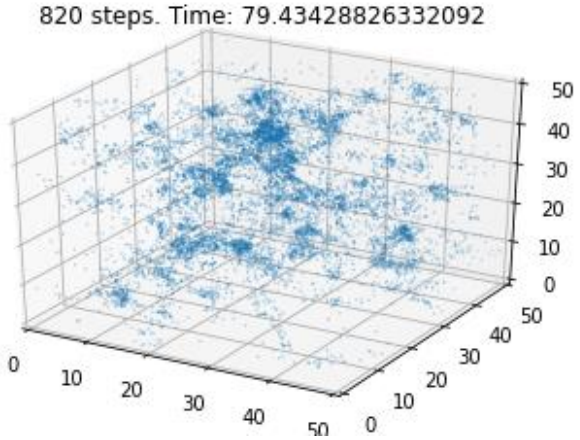


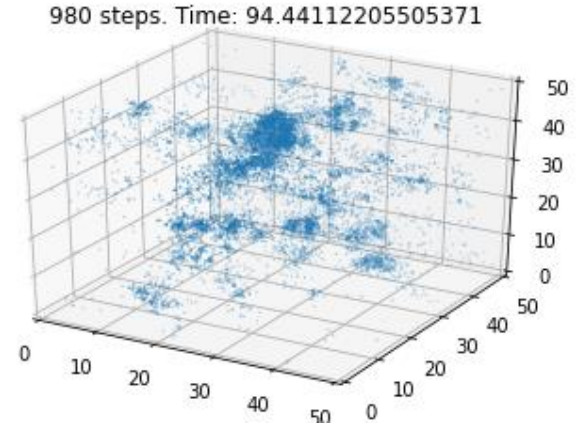
With periodic boundary condition:

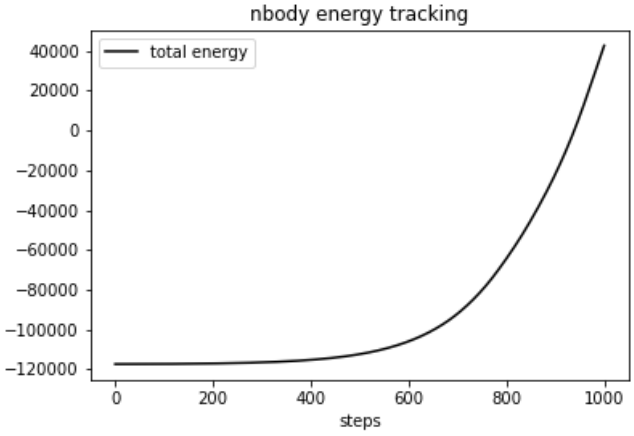





620 steps. Time: 60.72084856033325
0
10
20
30
40
50


820 steps. Time: 79.43428826332092
0
10
20
30
40
50


980 steps. Time: 94.44112205505371
0
10
20
30
40
50


nbody energy tracking
total energy
40000
20000
0
−20000
−40000
−60000
−80000
−100000
−120000
0
200
400
600
800
1000
steps

The particles forms clusters rather than a ball shape. The energy conserves better than the one with non-periodic condition. It blows up after about 600 steps because the particles are getting closer but the fixed timestep size is too large for them.

Task 4) Use masses derived from a realization of k^3

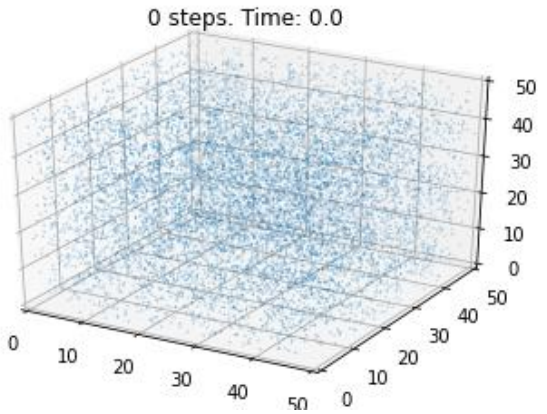
k^3 is a matrix in frequency domain. We need to transform it back to time domain to get the mass matrix. Then mass of each particle is assigned according to its position. Here is the function:

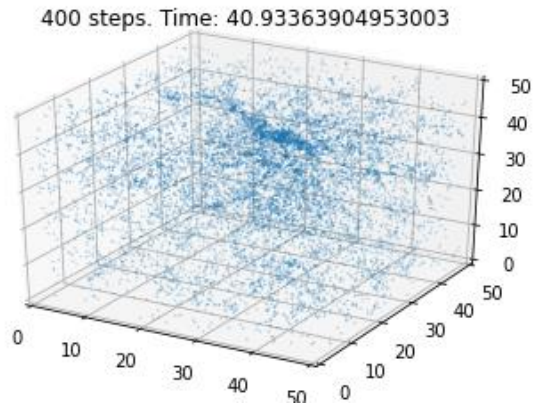
```
def k3_mass(den,edges,position,n,soft):

    x=np.arange(n)
    xmat,ymat,zmat = np.meshgrid(x, x, x)
    k = np.zeros([n,n,n])
    dr=np.sqrt(xmat**2+ymat**2+zmat**2+soft**2)
    k=1/dr**3

    new_den =np.fft.fftshift(np.fft.irfftn(k))

    ind_x = np.digitize(position[:,0],edges[0])-1
    ind_y = np.digitize(position[:,1],edges[1])-1
    ind_z = np.digitize(position[:,2],edges[2])-1

    new_m = new_den[ind_x,ind_y,ind_z]
    m_min = np.min(new_m)
    m = new_m/m_min
    return m
```

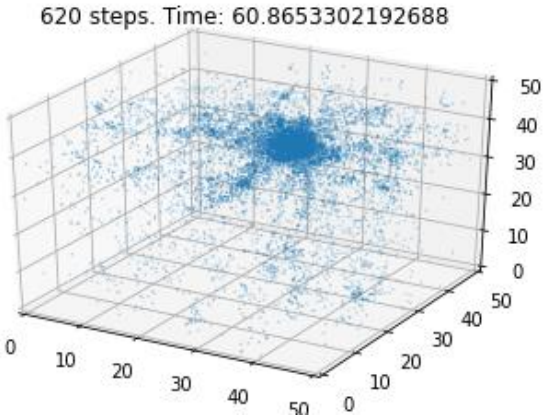
And here is how I updated the new mass:

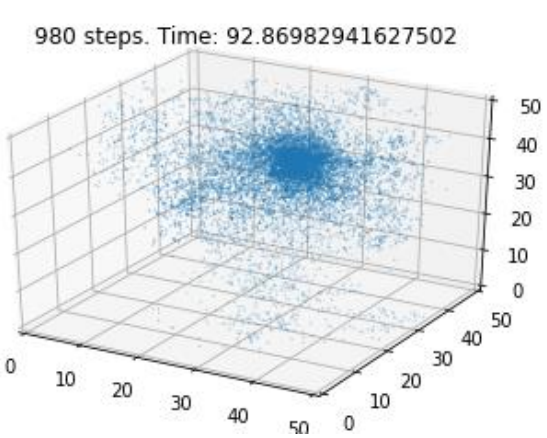
```
#derive masses from a realization of k^-3
den,edges = density(position, n, m)
new_m = k3_mass(den,edges,position,n,0.5)
```

The time evolution of my new universe looks like this:

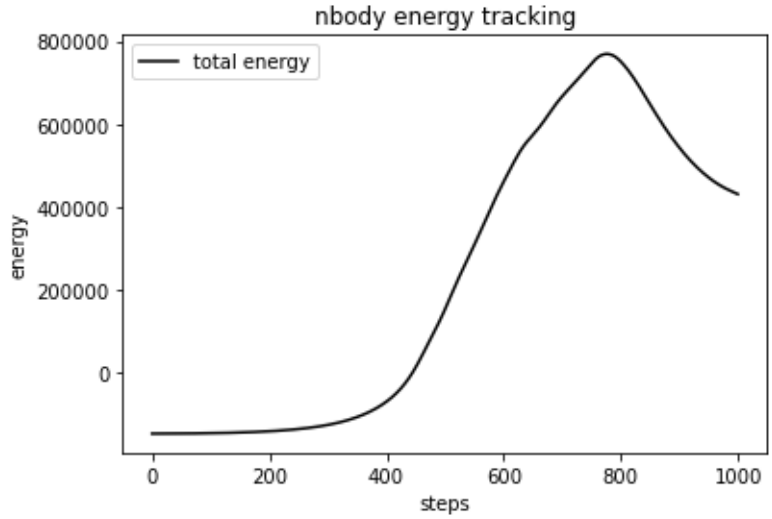








The particles now form a ball shape and the energy of the system stays conservative for some time and then blows up.



Here is a histogram of the masses derived from the k^3: